sgen4: A generator for small but difficult satisfiability instances

Ivor Spence
School of Electronics, Electrical Engineering and Computer Science
Queen's University Belfast
i.spence@qub.ac.uk

I. Introduction

This is a development of the previous generators sgen1 [1] and sgen2 [2] and generates both satisfiable and unsatisfiable instances. The instances are not derived solving any particular problem but are crafted to be as difficult as possible to solve. In all cases the variables in the generated instance are partitioned into small groups and for each group there are clauses to define relationships among the group. This is repeated for two or more partitions.

II. Unsatisfiable instances

Generating small yet difficult unsatisfiable instances requires a balance between the following constraints:

- To keep the instance short, each clause should eliminate a large number of possibilities.
- To make the instance hard to solve, the variables in each clause should not be "related", that is occur together in other clauses.

Unfortunately, these constraints conflict. Having unrelated variables tends to preclude a clause eliminating a large number of possible assignments. In the spirit of Hirsch's hgen8 program the compromise used here is to partition the variables into groups of size four and five (in two different partitions) and have multiple clauses re-use the variables in each group.

The basic technique here is identical to previous versions of the generator. In each partition one group contains five variables and the remaining groups each contain four. Thus the number of variables must be of the form $4g + 1$ where $g \in \mathbb{N}$. If the number of variables requested is not of this form the generator will use the next larger possibility - thus the generated instance will contain at least the requested number of variables, but may contain up to three more.

Within each group of four variables the idea is to permit at most two variables to be `false` For each group of size four, generate all possible 3-clauses of positive literals, i.e.

$$(a \vee b \vee c) \wedge (a \vee b \vee d) \wedge (a \vee c \vee d) \wedge (b \vee c \vee d)$$

This permits at most two variables from the set $\{a, b, c, d\}$ to be `false`. For the group of size five again generate all possible 3-clauses of positive literals (10 clauses), meaning that again only two variable from this group can be `false`. In total therefore, only $2(g-1)+2 = 2g = (n-1)/2$ variables can be `false`.

Now partition the variables into a different collection of $(g-1)$ groups of size four and one of size five and again for each group of variables generate all possible 3-clauses except this time use all negative literals. Thus now only $(n-1)/2$ variables can be `true`. Taking these two sets of clauses together it can be seen that it is not possible to assign a value to every variable since at most $(n-1)/2$ can be `true` and also at most $(n-1)/2$ can be `false`. Thus the generated instance is unsatisfiable.

If the number of variables is $4g + 1$ then the number of groups is g and the total number of clauses is $8g + 12$. Each clause has three literals, so if the number of variables is n, then as n increases the number of clauses is approximately $2n$ and the number of literals is approximately $6n$.

III. Satisfiable instances

To generate difficult satisfiable instances we use three partitions of the variables, all in groups of size five. Clauses generated from the first partition permit at most one variable from each group to be positive and clauses from the second and third partitions both require at least one variable per group to be positive. In principle therefore the instance may be satisfiable.

If the number of groups is g (so that $n = 5g$) then the instance will contain $10g$ binary clauses (from the first partition) and $2g$ 5-clauses (g from each of the second and third paritions), giving a total of $12g = 12n/5$ clauses and $6n$ literals (see Table I).

For each group in the first partition, we generate all possible binary clauses of negative literals (10 clauses for each group). This permits at most one `true` variable per group, that is a maximum of $g = n/5$ `true` variables overall.

For each group in the second and third partitions, we generate one 5-clause of all the positive literals. The $n/5$ `true` variable permitted by the first set of clauses might be enough to satisfy these subsequent clauses if they can be allocated as one per group.

If more and more collections of 5-clauses of positive literals are added it is less and less likely that the formula will remain satisfiable. Empirical results indicate that two collections give the most difficult instances for their size.

IV. Partitioning

For both satisfiable and unsatisfiable cases, to create difficult instances we need to ensure that there is as little connection as possible between the different partitions. For the first partition natural ordering is used, e.g. $\{1, 2, 3, 4\}, \{5, 6, 7, 8\}, \ldots$. Second and subsequent partitions are obtained by using simulated annealing, with a weight function which tries to minimise the extent to which the original partition is reflected in the second one. The difference between sgen4 and earlier versions lies in this function, which now tries to minimise the correlation between the second and third partitions as well as between first and second and between first and third. It is anticipated that this may make instances even harder to solve, but there is not yet sufficient empirical evidence to confirm this.

To ensure that satisfiable instances are created when requested, i.e. that it is possible to choose the g positive variables so that there is one per group in each of the three partitions, the technique used is to make an initial choice of the $n/5$ `true` variables and restrict the partitioning process to keep these variables in different groups. If the option -m model-file is chosen when generating a satisfiable instance, a satisfying model will be written to model-file.

Different partitions can be forced by using -s to specify the seed for the random number generation. If -s is omitted a value of 1 is used.

V. Parameters

The possible parameters are:

sat	Requests that a satisfiable instance be generated.
unsat	Requests that an unsatisfiable instance must be generated. Exactly one of `sat` and `unsat` must be specified.
reorder	Having generated the clauses as described up, a random permutation of variables and clauses is applied. This option is enabled by default.
n	Specify the minimum number of variables to be generated. Mandatory.
s	Specify a seed for random number generation. Defaults to 1.
m	Specify a filename for a satisfying model to be written to. Requires `sat` to be specified.

For example, typical invocations might be:

```
sgen4 -unsat -n 49 >u49.cnf
sgen4 -sat -n 120 -m s120.cnf >s120.cnf
```

VI. Results

Table I gives an example of a 9-variable unsatisfiable formula and a 10-variable satisfiable one.

Unsatisfiable (9 variables)	Satisfiable (10 variables)
p cnf 9 28	p cnf 10 24
-2 -3 -4 0	-1 -2 0
-1 -3 -4 0	-1 -3 0
-1 -2 -4 0	-1 -4 0
-1 -2 -3 0	-1 -5 0
-5 -6 -7 0	-2 -3 0
-5 -6 -8 0	-2 -4 0
-5 -6 -9 0	-2 -5 0
-5 -7 -8 0	-3 -4 0
-5 -7 -9 0	-3 -5 0
-5 -8 -9 0	-4 -5 0
-6 -7 -8 0	-6 -7 0
-6 -7 -9 0	-6 -8 0
-6 -8 -9 0	-6 -9 0
-7 -8 -9 0	-6 -10 0
4 2 6 0	-7 -8 0
7 2 6 0	-7 -9 0
7 4 6 0	-7 -10 0
7 4 2 0	-8 -9 0
3 8 5 0	-8 -10 0
3 8 9 0	-9 -10 0
3 8 1 0	9 6 3 5 1 0
3 5 9 0	4 2 8 10 7 0
3 5 1 0	7 5 3 9 2 0
3 9 1 0	10 4 8 1 6 0
8 5 9 0	
8 5 1 0	
8 9 1 0	
5 9 1 0	

TABLE I
EXAMPLE INSTANCES

References

[1] I. Spence, "sgen1: A generator of small but difficult satisfiability benchmarks," *J. Exp. Algorithmics*, vol. 15, pp. 1.2:1.1–1.2:1.15, mar 2010. [Online]. Available: http://doi.acm.org/10.1145/1671970.1671972

[2] A. V. Gelder and I. Spence, "Zero-one designs produce small hard sat instances," in *SAT*, ser. Lecture Notes in Computer Science, O. Strichman and S. Szeider, Eds., vol. 6175. Springer, 2010, pp. 388–397.